

# Posting with credibility in Micro-blogging systems using Digital Signatures and Watermarks: A case study on *Twitter*.

Georgios K. Pitsilis<sup>1</sup> and Mohamed El-Hadedy<sup>2</sup>

<sup>1</sup> Computer Science Research, Athens, Greece  
georgios.pitsilis@gmail.com

<sup>2</sup> Coordinated Science Lab, Urbana, Illinois, USA  
hadedy@illinois.edu

**Abstract.** Micro-blogs are contemporary broadcasting services, for exchanging small elements of content, including video and images. Despite its popularity, micro-blogging is not without issues. So far, various security concerns, such as: privacy and confidentiality of micro-blogging systems have attracted the interest of the scientific community. Nevertheless, in this document we refer to a security issue that is concerned with the posting and circulation of fake messages. With the aim to make micro-blogs a credible source of information, in this work we propose a protocol, as a solution, that can be easily adapted to existing services such as: *Twitter* and *Facebook*. With the main keys of the solution being the employment of digital signatures and watermarks, the protocol can serve postings of various types of content. Finally, the solution serves the dual purpose of proofing the fake messages as well as the repudiation of postings.

**Keywords:** Twitter, Micro-blogging, Security, Digital Signatures, Digital Watermarking, Hoax Tweets

## 1 Introduction

Social micro-blogging systems are increasingly popular. Unlike the traditional blogs, they allow users to exchange small elements such as short sentences (called micro-posts), which is the major reason why they received wide adoption by the users. Normally, micro-posts are forwarded in emails and other distribution services around the media and are used as a means for spreading the news. It is quite common for micro-blogs to allow the inclusion of images and video links for making the posts more engaging. Despite the privacy features offered by micro-blog services, aiming to provide full control on the content published by them, there are still serious vulnerabilities. For instance, *Twitter* and *Facebook* are exposed to threats related to the malicious reproduction of news, or the unauthorized re-posting of the visual material [9,10,8], with both users' and services' reputation strongly being affected [8].

On the other hand, there is no mechanism on the social media for tracking any intentionally withdrawn postings of a user. The lacking of mechanisms to apply such restriction has rendered micro-blogs unsuitable for the posting of highly important statements. In *Twitter*, any Internet user can register with the service, by simply choosing a user-name for identification, along with a valid email address. Then, she can freely post short messages (up to 150 characters long), which will be publicly exposed via her personal account page. Meanwhile, other *Twitter* users can register as followers to her, so that they can refer to the posts. Followers are also permitted to re-tweet messages of their followers, in which way, a reference to the first message would re-appear in the follower's account page.

The incentives for attacking a Micro-blogging system can be varying. The provocation of legitimate users, the acquisition of benefits from unauthorized use of the digital material, or just the spread of panic [5], can be some of them. With respect to the technique used, attacks can be classified into: *Hack tweets* and *Hoax tweets*. The former regards tweets sent from a hacked legitimate account, while the latter is concerned with the posting of snapshots of fake tweets to various websites. (See example in fig.1). As far as the former type, an incident known as *Hack crash* [6], shows the level of damage that can be caused by such attacks. In *Hack crash*, tweets were sent from a hacked *Twitter* account resulted to short-time panic in the US stock market in April 2013 [1]. The use of automated financial algorithms in stock market systems, which retrieve information from social media to predict future trends of the market [11], was mainly responsible for the panic caused. In that incident, the extensive automation in the algorithms used, as they were reacting quicker than human thought, gave no time for the involved parties to discover the nature of tweets.

As far as the *Hoax tweet* type, a threat can be simply thought of as a malice case of user creating fake tweet content, claimed as being posted by the attacked account. The publishing of visuals alongside tweets, with text not tightly attached to the images, offers space for persuasive *hoax-tweets*. The availability of many on-line services for creating fake tweets of any content [13,12,3] is a factor for encouraging attackers to target any legitimate *Twitter* user. It is worth mention, the existence of various toolkits, such as *Twitt-embeds* by *Twitter* API [14], which provide seamless embedding of tweets into a web page, and which could be potentially used for malicious purposes. The output of *Twitt-embeds*, by just being an ordinary image file, can be accessed by any user without necessarily being a *Twitter* user. That could be a potential vulnerability for the *Twitter* service itself, because since it can not be resolved if an embedded tweet that looks original, is truly posted or faked.

For *Facebook*, the unauthorized reposting of images already posted by legitimate users [10], is the main threat for the service itself, as there is lack of protection for media distribution. The similarities between the nature of the services and the security policies used in *Facebook* and *Twitter*, suggests applying some generic solution.



**Fig. 1.** Example of fake tweet. (Ref. [9])



**Fig. 2.** Example of fake picture of stormy New York skyline used in *Twitter* for hurricane Sandy. (Ref. [7])

## 2 Existing approaches

*Twitter* does not employ any security mechanism to avoid the above incidents. For *Twitter*, the only means of protection so far for the account owners, regards restrictions that can be provisionally applied to followers, allowing only those approved by the account owner to access the posted material. Nevertheless, such policy, in the way it works, has the result on tweets visibility seriously being affected. Other approaches include classification techniques which are known to achieve some success on detecting images that look irrelevant with the posted text on *Twitter* [7]. (see fig. 2.)

While the application of higher security measures on the existing user authentication system could, at some point, prohibit *Hack tweets*, on the other hand, the prevention of *Hoax tweets* requires a different approach, which we elaborate in this document.

## 3 The proposed solution

The solution we propose focuses on the issues of *Hoax-tweets* and it provides two alternative schemes that combine: Watermarking, One-way functions and Public Key encryption. The main idea is that, the posted material, prior to publishing in the micro-blogging system, would first get digitally signed and uniquely tagged with appropriate evidence to prove its origin.

Considering *Twitter* as a pilot use case, with tweets consisting of text alone, a likely solution would be to apply hashing and pair-key cryptography. Hashing

is applied onto the text content to shorten its size. This is quite necessary for messages of long size. To create a unique ID for the message, the hash value is signed with the user's Private key. For the case of tweets comprising images, we propose the solution of embedding the signatures into the images themselves in the form of watermarks. To enhance robustness, the tweet signatures could include a digest of the attached image.

Next, we explain in more detail the protocol that implements the proposed scheme, as well as some variations of it. All the schemes we propose require the existence of a Trusted Third Party (TTP), in which, users who wish to take advantage of the protection scheme, will first need to register with a Private key. (See algorithm 1.) TTP could also serve as the entity to perform the watermarking service. In such scheme, the resolution, that is the verification of authenticity of some post, can be performed by any entity with the provision of the Public key of the tweet's owner.

The following symbols have been used: The entity  $A$  denote as the owner of the *Twitter* account and  $B$  is the entity serving as TTP.  $K_A$  is the Private key of  $A$ , while  $K_{PA}$  is his Public Key.  $B$  could also run on the client side ( user's hardware), without any compromise in the security.

---

**Algorithm 1:** Registration with TTP

---

COMMUNICATION	COMMENT
1: $A \rightarrow B : K_A$	Compute and send Private key to trusted party $B$
2: $A \rightarrow B : K_{PA}$	Compute and send Public key to trusted party $B$

---

### 3.1 For tweets containing text alone

For tweets consisted of text alone, prior to publication to *Twitter* service  $C$ , the text content  $M$  would be first hashed (  $\#M$  ), before it gets signed with the private key  $K_A$  of user  $A$ . (See algorithm 2). The final tweet includes the text message (step 2), plus a unique ID, which we call the *key-code* of the tweet. That code will be shown publicly, along with the posted text (step 3).

---

**Algorithm 2:** Tweeting text alone

---

COMMUNICATION	COMMENT
1: $A \rightarrow B : M$	Send text message $M$ to trusted party $B$
2: $B \rightarrow C : M$	Forward text message $M$ to <i>Twitter</i> service $C$
3: $B \rightarrow C : \{\#(M)\}_{K_A}$	<i>key-code</i> posted along the hashed tweet text $M$

---

For verifying the authenticity of text tweets, we include another entity  $D$  in the scheme, acting as someone wishing to verify the origin of some tweet. (See

algorithm 3). There is no requirement in the scheme for  $D$  to be a trusted party. As such, it is reasonable to assume that  $D$  should be allowed to use as input, the publicly shown text content  $M$  of a tweet, along with the *key-code*  $\{\#(M)\}_{K_A}$  computed for that content. Then, by decrypting the *key-code*, the signature of the hash value is received (step 2). Next,  $D$  would hash the twitted plain-text  $M$  to get the signature of it (step 3), which he would then compare with the output of the previous step (2), to verify the origin of the tweet.

---

**Algorithm 3:** Verification of text-based tweets

---

COMMUNICATION	COMMENT
1: $B \rightarrow D : K_{PA}$	Public key of $A$
2: $C \rightarrow D : \{\{\#(M)\}_{K_A}\}_{K_{PA}}$	Verifying the originality of the <i>key-code</i>
3: $C \rightarrow D : M$	by comparing the output with the plain text $M$

---

### 3.2 For images posted alongside text tweets

Images posted alongside text, should be first watermarked with the signed *key-code* ID of the text content, as outlined in the previous step. That is necessary for creating a connection between the textual and the media component of a post. The process is shown in algorithm 4 and is detailed as follows: The expression  $wm(I, P)$  in step 3 denotes the operation of hiding the string  $I$  into picture  $P$  by watermarking, which in our case, the picture is the image the user wishes to post alongside a text tweet. As before, the watermarking task can be carried out by the trusted entity  $B$ . The scheme allows the posting of more than one image alongside a text tweet, which can be achieved with the repetition of step 3 for each additional image.

---

**Algorithm 4:** Attaching an image to a tweet

---

COMMUNICATION	COMMENT
1: $A \rightarrow B : \{\#(M)\}_{K_A}$	The <i>key-code</i> is computed and submitted to the trusted party $B$
2: $A \rightarrow B : P$	The picture $P$ is sent to the trusted party $B$
3: $B \rightarrow C : wm(\{\#(M)\}_{K_A}, P)$	Embedding the encrypted hash value of $M$ into the watermarked picture
4: $B \rightarrow C : \{\#(M)\}_{K_A}$	The <i>key-code</i> is posted to <i>Twitter</i> service $C$ alongside the watermarked picture
5: $B \rightarrow C : M$	The text is finally posted to <i>Twitter</i> service $C$

---

In the verification shown in algorithm 5, the expression  $exw(P_{pub}, V)$  denote as the function used for extracting a watermark with content  $V$  from the watermarked picture  $P_{pub}$ . This function returns either  $V$  or null in case the content  $V$

has not been detected into  $P_{pub}$ . The assumption made is that, extracting a watermark actually means detecting the existence of a particular sequence of bytes within an image. This has been found to be a good practice for watermarks used for serving a security purpose [4]. Therefore, the verification scheme we present can only serve the case when only requiring to answer the question, whether a picture has been posted along with a text tweet by some user  $X$  or not. Such verification scheme also provides the benefit of not needing to store the original, non-watermarked picture somewhere in the system, thus enhancing the security of the scheme.

---

**Algorithm 5:** Verification of pictured tweets

---

COMMUNICATION	COMMENT
1: $C \rightarrow D : M$	The message in plain text as posted
2: $B \rightarrow D : K_{PA}$	The Public key for decryption
3: $C \rightarrow D : \{\#(M)\}_{K_A}$	The <i>key-code</i> of the <i>Twitter</i> posting
4: $C \rightarrow D : \{exw(P_{pub}, \#(M)_{K_A})\}_{K_{AP}}$	Extracting and decrypting the info to verify the <i>key-code</i>

---

### 3.3 Improved scheme for higher robustness

For improved security we propose an alternative protocol, which provides over the simplified version in section 3.2, a secure link between the tweeted text and any pictures attached to it. In addition to the simplified version, the tweet's *key-code* ID is derived from both the image content and the text tweet. Furthermore, a time-stamp has been included in the *key-code*. That was for securing the protocol against likely attacks in which, text and images from various tweets posted by the same legitimate user  $A$  in the past, are mixed together by an attacker to produce a hoax-tweet.

For example, lets assume the scenario in which a legitimate user has posted a tweet at time T1, and is composed of a text message M1 and a picture P1. The same user, at time T2 posts another message with text content M2 that is exactly the same with M1, but this time attaching picture P2 to it. If algorithm 4 was used, an attacker would still be able to violate the security of the system, by creating a new hoax tweet at time T3, composed of the textual content T1 attached to picture P2.

Despite both text and image have been posted by the same legitimate user, an attacker can still create a hoax one, by mixing the components across tweets. More important, the hoax tweet yet would look original, because there is no way in the former scheme to prove that such tweet can be the product of tampering. The reason is the use of unaltered components of legitimate tweets to make the hoax tweet.

Likewise, for tweets produced by their account owners, there is no way to oppose a complainant's claim that such tweets are not original. We attribute the reason for the above issues, to the fact that, in the above design the components of a tweet cannot be uniquely referred.

The general idea of an alternative protocol to tackle the above issue can be described as follows: The *key-code* ID is composed of two parts, concatenated together: The first part contains the ID provided by hashing the text message with the time-stamp, while the second part can be a unique value, associated with the attached picture. As opposed to the simple version of the protocol in algorithm 4, in this one, the time-stamp was included for ensuring the uniqueness of the composed *key-code* ID. In this way, likely collisions can be prevented, that may occur when attempted the use of the same *key-code* in two different tweets. Furthermore, the scheme can make provable as well, if some tweet text or some picture have ever been posted before.

The protocol is detailed in algorithm 6 and it works as follows: A digest of the image is produced by hashing its file content, in step 3. Then, the produced unique ID is used along with the text message  $M$ , which is also hashed for signing the tweet.

In the notation used, the plus symbol (  $+$  ) denotes concatenation.  $T$  is used to denote a time-stamp. For short, we call  $P_{pub} = wm([\#(M + T)]_{K_A}, P)$  the publicly shown watermarked picture that is attached to the tweet, while  $P$  denote as the original picture.

---

**Algorithm 6:** Attaching images on a tweet. The provable way

---

COMMUNICATION	COMMENT
1: $A \rightarrow B : P, M, T$	The message $M$ , the time-stamp $T$ and the picture $P$ are send over to the trusted third party $B$ .
2: $B \rightarrow C : wm([\#(M + T)]_{K_A}, P)$	The watermarking of time-stamped message into the picture, is performed by the trusted party.
3: $B \rightarrow C : \{\#(M + T)\}_{K_A} + \{\#(P_{pub})\}_{K_A}$	The concatenation of $M + T$ with the picture, provides a unique <i>key-code</i> to display, composed of two segments.
4: $B \rightarrow C : M + T$	The message $M$ and time-stamp $T$ are posted along with the <i>key-code</i> .

---

In step 2, the digest of the time-stamped message content is embedded into the watermark, as a way to uniquely referring to it. In step 3, the unique tweet ID (*key-code*) is derived from both the picture and the text message content. In addition,  $T$  has been included to prevent attacks, in which a malicious user could create a hoax tweet by mixing various components from past tweets.

If required to attach multiple pictures to a tweet, the 3rd step can be modified accordingly to accommodate the signatures of all pictures together. As such, step 3 should be repeated for every picture to be attached.

---

**Algorithm 7:** Verification of provable pictured tweets

---

COMMUNICATION	COMMENT
1: $B \rightarrow D : K_{PA}$	The public key of $A$
2: $C \rightarrow D : M + T$	The message timestamped
3: $C \rightarrow D : \{\#(M + T)\}_{K_A} + \{\#(P_{pub})\}_{K_A}$	The <i>key-code</i> of the posted tweet is provided as public information.
4: $C \rightarrow D : \{exw(P_{pub}, \{\#(M + T)\}_{K_A})\}_{K_{PA}}$	Extracting, decrypting and verifying the text message from the picture.
5: $C \rightarrow D : \{\{\#(M + T)\}_{K_A}\}_{K_{PA}} + \{\{\#(P_{pub})\}_{K_A}\}_{K_{PA}}$	Verifying the originality of the <i>key-code</i> .

---

As previously mentioned, the verification serves a dual purpose. First it is to verify that some picture has been posted along with a particular tweet (step 4), and second, to prove the originality of the composed tweet as a whole (step 5). In step 4, the first part alone of the *key-code*  $\{\#(M + T)\}_{K_A}$ , suffices to confirm that the picture  $P$  has been truly attached by the legitimate user in that tweet. The verification of the originality of the *key-code* is performed in step 5, where we used as input the watermarked picture  $P_{pub}$  along with the timestamped plain message  $M + T$ . In this way can be proved the originality of a tweet, by checking whether the *key-code* can be successfully re-composed from these 3 elements. That is done by decrypting each half of the key-code in step 5, and comparing it with its adjacent part from step 3.

## 4 Implementation for the *Twitter* service

As a proof of concept, we implemented a service that follows the above design. The provided UI allows to any user already having a *twitter* account to use his credentials for signing up to this service and start using it. As such, any new tweets posted by this user via our UI, will become securely signed before they get posted on *Twitter*. (See fig.3) Furthermore, all the evidence required for the verification of tweets, become publicly available in our service's web site. The service also features a publicly accessible searching and browsing tool, for finding tweets posted by any registered user. (See fig.4)

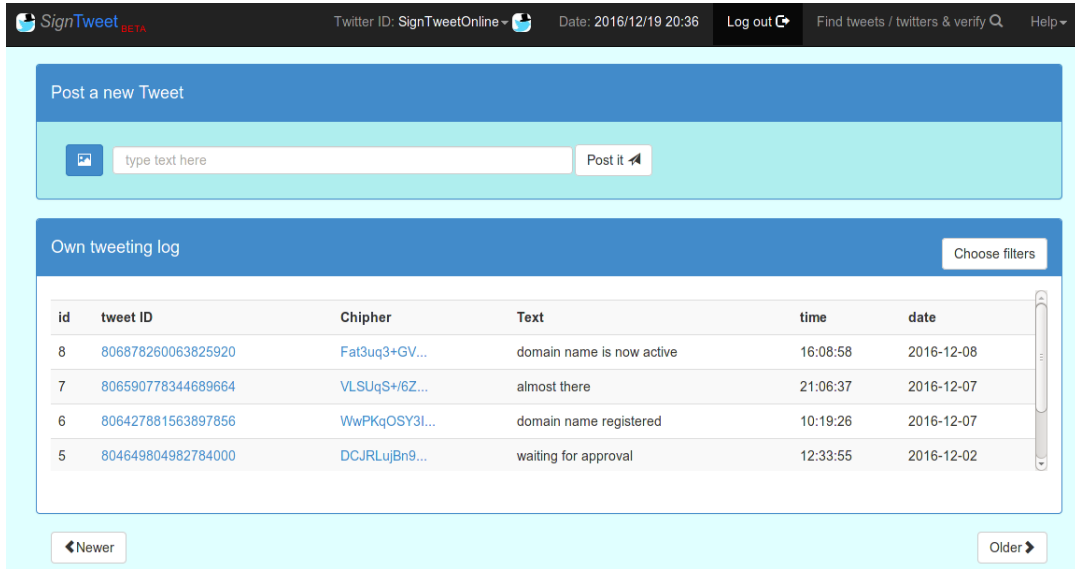
The status of the current implementation is: "work in progress", therefore it currently supports only a subset of the full functionality. More particularly, only text tweets can be submitted and verified. We left the watermarking feature of any attached pictures as a future work.



We should also point out that, for this implementation, due to the short message length allowed in the *Twitter* service, we found unnecessary to apply hashing onto the text messages (as per step 3 in algorithm 2). That is because hashing very short messages results to lengthy cipher-text, which is not practical for *Twitter*.

The following software tools were used in the implementation : *PHP phpseclib* [2], which contains the *RSA encryption library*, and the *base64-encode & base64-decode* modules. For accessing the users data and services offered by the *Twitter* platform through program code, we used an API, especially built by Abraham [15]. The above API was chosen as it can work with PHP server-side scripting language and functions as REST interface.

To avoid any misinterpretation of the information by the communication protocols during the transmission across the network, we used 64-base encoding for the storing of cipher-text and encryption keys in the system.



**Fig. 3.** The user interface for posting tweets

To eliminate any doubts regarding the validity of the verification, as can be seen in fig. 5, the result is presented along with all the information necessary for anyone to be able to re-validate this step by other means. In that respect, the employment of external tools with same functionality should be sufficient to check the validation of the process. For instance, by applying RSA decryption onto the *Tweeted Text Cipher* using the presented *Public Key* is the way to perform custom verification. As per algorithm 3, showing a decryption output that

Search the database of signed tweets to verify

name  Sign  posted within  Find

Tweets for verification

User	signTweet ID	tweet ID	Text	Chipher	Date	Time	
SignTweetOnline	8	806878260063825920	domain name is now active	Fat3uq3+GV...	16:08:58	2016-12-08	Verify <input type="button" value="✓"/>
SignTweetOnline	6	806427881563897856	domain name registered	WwPKqOSY3l...	10:19:26	2016-12-07	Verify <input type="button" value="✓"/>

◀ Newer Older ▶

**Fig. 4.** The user interface for searching and verifying tweets.

equals to the displayed *Tweeted Text*, would be enough to prove the originality of the posted message.

Our service currently serves experimental purposes only and it is totally free of charge. The service can be reached at the web address: <http://www.signtweet.com>.

Next, we present a likely scenario of use:

1. Registration. Users who have not been registered with our systems, can do so by using their *Twitter* credentials. Upon registration, both the Private and Public keys are automatically computed by our system, and are allocated to the new user.
2. Posting a tweet. Upon submitting a text tweet using the signTweet GUI (See fig. 3), a cipher-text is computed based upon the Private key, and it is presented in the user's history page, along with the plain-text. Afterwards, the textual content is automatically forwarded and displayed in the user's official page in *Twitter*.
3. Verification. This step is optional, it can be run by anyone, hence not requiring registration. The purpose served is to provide the means to verify the originality of any tweets posted through our service. Using our UI, any tweet posted via our service can be retrieved by applying the proper criteria. (See fig. 4.) Next, by clicking the *Verify* button for this tweet, in a separate window is shown all the evidence required for the verification, such as: the Public key of the posting user, the cipher-text of the tweet as well as the plain text. (See fig.5). Finally, by clicking the button created for this purpose, and it is called "Vefiry this tweet", the system proceeds to the verification based

Verification of tweet with ID: **806878260063825920**

Twitting user: [SignTweetOnline](#)

Public key of the tweeting user (base64 encoded)

```
LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0Nck1JR2ZNNQTBHQ1NnR1NjYjN
EUUVQVVFVQUE0R05BRENCaVFLQmdRQ2kxbVJNMmUwazRPM0Z
kK0FZbG11eVptbi8NCiFycDNYa0NlSnNjWlZiR040SVd3RjNoVjA5MEZ0eVJ
Ca1c2YTlacWVxUmFyQ0ljdWRqcUVCQ29jWDV6REgzVFQNCkFIYzk2eY
zMxdQcEdOME5EVGRjTnV3UldUWGI3c29EUxVUNFhQVUR2dEI4bmh1aT
dxb3hsSUJoOGRVUzJ1MnYnClvZ0E4Ri82eGc4MTc2NEp1d0IEQVFBQg0KL
```

Public key of the tweeting user (plain)

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsGSIb3DQEBAQUAA4GNADCBiQKBgQC1mRM2e0k4O3Fd
+AY1muyZmn/
Qrp3XkCeJscYyYGN4lWwF3hV090FtyRBkW6a9ZqeqRarCB2udjqEBcocX5zD
H3TT
AHc96zV31wPpGN0NDTdcNuwRWTXiwsODQun4XjQTvtB8nhui7qoxlBh8d
```

Tweet ID: **806878260063825920** ( on twitter )

Tweeted text

domain name is now active

Tweeted text chipher (RSA encryption)

```
Fat3uq3+GV8/hVp4da6k1OBE2ghl3gD9teyUj9CGyoG
+1X8QORtIAUFdorXoXbBMhVSI3Q8K9S1wVs3nOlPolCLw+HOzxtQstxoY/
KMYizivOys+TjX9/qusZEy0czVHwDMR7OYnekBtebK/USnHBueO59WK7w/
```

Verify this tweet ☒

Tweet verification successful.

**Fig. 5.** Tweet verification result

upon the above data. It is worth mention that, the data in the verification window are in editable form, so that the user can examine the verification output in an interactive way, by optionally modifying the evidence.

Future work in our implementation schedule includes, the support of the protocol for other forms of posting material, such as photos, as per algorithms 4,5,6 and 7.

## 5 Discussion

The proposed design can successfully serve a dual purpose as we described previously. In one use-case, it can serve the purpose of account owners wishing to prove that any claims by other users about a posted tweet is truly a hoax [9]. In another use-case, in which is claimed that offensive tweets have been publicly posted, but removed afterwards by the legitimate account owner, the proposed scheme can make possible the verification of such claim. Whoever has retained a snapshot of the tweet in question along with the cipher-text, can certainly prove

his claim that such tweet has ever been posted by the legitimate account owner. In either case, the assumption is, the embedding of such service into micro-blogs, will be for the benefit of whoever is operating at good practice. (either a legitimate *Twitter* account owner, or a complainant). In that way, potential attackers would be discouraged from producing fake tweets for any user who has registered to our service. Similarly, a legitimate *Twitter* user, by adapting such protection scheme, could easily build a reputation of behaving in good practice.

## 6 Conclusion

In this document we provided an algorithm that could help to make micro-blogging systems more credible and secure. It can support micro-posts of multiple types, such as text alone, or text with attached images. We provided algorithms for securing and verifying the authenticity of posted messages using Public Key Encryption, Digital Signatures and Watermarking. We also presented a proof-of-concept implementation of the above algorithm in the form of a publicly accessible service. We are hoping such protocol to receive great adoption by the existing micro-blogging services.

## 7 Acknowledgements

We would like to thank *Prof. Kevin Skadron* from the Department of Computer Science, School of Engineering and Applied Science, University of Virginia, US, for his contribution and the useful discussions on the subject.

## References

1. "AP Twitter hack causes panic on Wall Street and sends Dow plunging". <http://www.theguardian.com/business/2013/apr/23/ap-tweet-hack-wall-street-freefall>, 2013. Accessed: 23.04.2013.
2. PHP Secure Communications Library. <http://phpseclib.sourceforge.net>, 2015. Accessed: 5.11.2016.
3. S. Demian. Fake tweets generator. <http://www.lemmetweetthatforyou.com>, 2012. Accessed: 5.11.2016.
4. M. El-Hadedy, G. Pitsilis, and S. J. Knapkog. An efficient authorship protection scheme for shared multimedia content. *Image and Graphics, International Conference on*, 0:914–919, 2011.
5. N. Feeney. "Fake Twitter Bomb Threats to Airlines On the Rise". <http://time.com/3685595/airline-security-twitter-threats/>, 2015. Accessed: 28.01.2015.
6. B. Gambini. "With one false tweet, computer-based Hack Crash led to real panic". <http://www.buffalo.edu/news/releases/2015/05/037.html>, 2015. Accessed: 20.05.2015.
7. A. Gupta, H. Lamba, P. Kumaraguru, and A. Joshi. Faking sandy: Characterizing and identifying fake images on twitter during hurricane sandy. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, pages 729–736, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

8. E. Hannon. Twitter CEO Admits: "We Suck at Dealing With Abuse and We've Sucked at It for Years.". [http://www.slate.com/blogs/the\\_slatest/2015/02/04/twitter\\_ceo\\_dick\\_costolo\\_admits\\_failure\\_stopping\\_abuse.html](http://www.slate.com/blogs/the_slatest/2015/02/04/twitter_ceo_dick_costolo_admits_failure_stopping_abuse.html), 2015. Accessed: 04.02.2015.
9. Huzlers.com. "Buzz Aldrin Admits Apollo 11 Moon Landings Were FAKE and Simply A Set". <http://huzlers.com/buzz-aldrin-admits-apollo-11-moon-landings-fake-simply-set-see-tweet/>, 2014. Accessed: 12.11.2014.
10. J. O'Neill. "The Disturbing Facebook Trend of Stolen Kids Photos". <https://www.yahoo.com/parenting/mom-my-son-was-digitally-kidnapped-what-112545291567.html>, 2015. Accessed: 03.03.2015.
11. P. Papaioannou, L. Russo, G. Papaioannou, and C. I. Siettos. Can social microblogging be used to forecast intraday exchange rates? *Netnomics*, 14(1-2):47–68, Nov. 2013.
12. Simitator.com. Facebook status generator. <http://simitator.com>, 2015. Accessed: 5.11.2016.
13. tweetfake.com. Fake tweets generator. <http://tweetfake.com>, 2015. Accessed: 5.11.2016.
14. Twitter-Inc. Embed a single tweet. <https://dev.twitter.com/web/embedded-tweets>, 2015.
15. A. Williams. TwitterOAuth. "The most popular PHP library for use with the Twitter OAuth REST API". <https://twitteroauth.com>, 2015. Accessed: 5.11.2016.